Electronic Communications of the EASST
Volume 076 (2019)

Automated Verification of Critical Systems 2018 (AVoCS 2018)

A Framework for the Formal Verification of Networks of Railway Interlockings - Application to the Belgian Railway

Christophe Limbrée, Charles Pecheur

17 pages

Guest Editors: David Pichardie, Mihaela Sighireanu
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A Framework for the Formal Verification of Networks of Railway Interlockings - Application to the Belgian Railway

Christophe Limbrée[1], **Charles Pecheur**[1]

christophe.limbree|charles.pecheur@uclouvain.be
Université catholique de Louvain, Louvain-La-Neuve, Belgium

Abstract: Modern railway stations are controlled by computerized systems called interlockings. In fact the middle size and large size stations usually require to use several interlockings, then forming a network of interlockings. Much research propose to verify the safety properties of such systems by means of model checking. Our approach goes a step further and proposes a method to extend the verification process to a network of interlockings. This process is known as compositional verification. Each interlocking is seen as the component of a larger system (i.e., station) and interacts with its neighbours by means of interfaces. Our first contribution comes in the form of a catalogue of elements that constitute the interfaces (as used in the Belgian railways) and associated contracts. Each interface can then be bound to a formal contract allowing its verification by the OCRA tool. Our second contribution comes in the form of an algorithm designed to split the topology controlled by a single interlocking into components. The verification of a large station can therefore be achieved by verifying its constituting components and their interaction thereby tackling the state space explosion problem while providing guarantees on the whole interlocking.

Keywords: Interlocking, Model checking, Compositional verification, OCRA, NUXMV, PDR, ic3

1 Introduction - Interlocking system

In the railway domain, an interlocking is an arrangement of systems that prevents conflicting train movements in a station. The main component of an interlocking is a safety-critical system relying on a generic software and application data in order to control the railway traffic in a station. Figure 1.1 illustrates a toy station controlled by an interlocking.

The application data are the configuration layer needed by the interlocking in order to control a specific station. In a route-based interlocking, the application data are organized in routes. For example, the route R_C_043 is a path in the station shown in Figure 1.1 granting passage to a train from signal C to track 043. The main elements/resources used by a route are: the points, the track circuits, the track, and the signals. A point is a mechanical installation enabling railway trains to be guided from one track to another. A track circuit is a simple electrical device used to detect the absence of a train on rail tracks. A track is a position on railway network where a route starts or ends. A signal is an electrical device used to pass information relating to the state of the route ahead to train drivers. A route takes different states: it goes from *not set* to *set*, to

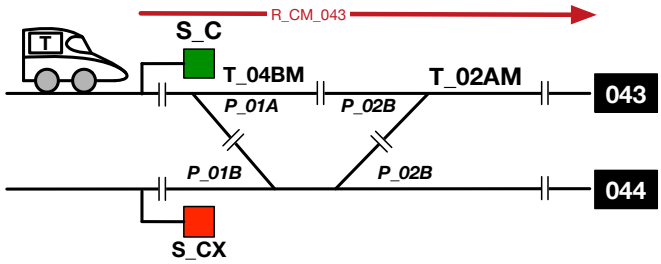

Figure 1.1: INTERLOCKING - A first toy example of station.

proved, and returns to *not set*. In "proved" state, the train is allowed to proceed on the route as the route origin signal shows green aspect. The resources allocated by the route are freed when the train runs through the route according to a specific sequence. The route is then released.

```
1 *Q_R(C_043)
2   if  R_C_043 xs
3     P_01A cfn, P_01B cfn, P_02AB cfr, P_02B cfr
4     U_IR(01A) f , U_IR(02B) f
5   then  R_C_043 s
6     P_01A cn , P_01B cn , P_02A cr , P_02B cr ,
7     U_IR(01A)  l , U_IR(02B) l
8     U_CBSPA(043) l
```

Listing 1: Example of application data - route - first step.

Listing 1 shows an extract from the application data corresponding to the control of route **Q_R(C_043)** (line 1). The first part of the code holds the conditions that allow the route control (line 2 - 4). First the route is checked not being set yet (line 2). Second (line 3) the points are checked to be movable to the position required by the route. For example, the point P_01A must be **cfn** meaning free to move to normal position. The last line (4) refers to the locking of the points. The second parts of the listing enumerates all the actions taken if the conditions are fulfilled. Line 6, the points are controlled (e.g., P_01A is controlled left *cn*). Second, the points locking variables are put in locked state (line 7). Line 8 refers to the directional locking. Indeed track 043 can be accessed from the left and from the right. Thereby in order to avoid collisions, the interlocking shall not grant two routes in opposite direction to 043 at the same time. That's the purpose of the **U_CBSP** variable where the suffix **A** is used from left to right and the suffix **B** in the opposite direction.

In our first paper [BCL+15], we presented the model-checking of the interlocking of a small station of the Belgian network (Namêche) by translating the SSI (Solid State Interlocking) application data into a NUSMV model. In our case study, the BDD (Binary Decision Diagram) algorithm provided in NUSMV easily allowed to verify invariants whereas a special algorithm using PyNuSMV [BP13] was used to verify CTL formulas (Computation Tree Logic). Considering the fact that a verification process based on a monolithic model would not scale up to stations of larger size, we proposed a novel method based on compositional verification [Lim16]. This approach used a compositional verification framework called OCRA [CDT13, CT12, ocr18]. All the configuration (components and contracts) was done manually as opposed to our new contribution

where our algorithm retrieves both the components and contracts and produces the configuration files for OCRA.

In Section 2 we provide the background for the compositional verification. In Section 3, we introduce our catalogue of interface and contract definitions. In Section 4, we propose an algorithm designed to split a station into smaller components in order to tackle the space state explosion problem. In our case study in Section 5, we apply our algorithm and catalogue in order to verify the "no-collision" safety property on a realistic size station of the Belgian network. Finally we conclude in Section 6.

1.1 Validation of interlocking

The application data are prepared manually and are therefore subject to human errors. For example: conditions can be missing in a route control, such as a point condition. This kind of error could easily be discovered by a **code review** or by **testing** on a simulator (e.g., route by route). A much harder to find and malicious problem is that caused by **concurrent actions** (e.g., route or point requests). In this case, the combination of possible concurrent actions explodes quickly and testing all possible combinations manually is not tractable.

Currently the validation of the application data relies on the combination of different V&V activities such as functional tests, safety tests, and application data reviewing. The functional tests ensure that the system responds properly to the commands issued by the traffic controller. During safety tests, all the conditions that are supposed to impact the routes are tested in their restrictive states (e.g., a point in the wrong position shall not allow the origin signal of a route to go green). Those tests are prepared and carried out by an independent tester. Finally, the **application data are reviewed** by the technical manager in charge of the project. All anomalies are traced in a bug management tool and the engineer in charge ensures that they are all closed before the interlocking is commissioned.

1.2 Formal verification of interlocking

In their paper [FFM12], Fantechi *et al.* clearly states that the future of model checking for interlocking lays in the SAT solvers which is the direction that we have taken in our research by using the PDR/ic3 (Property Directed Reachability/Incremental Construction of Inductive Clauses for Indubitable Correctness) algorithm in the NUXMV model checker [Rob14]. Recently Fantechi *et al.* [A. 17] proposed a similar approach based on compositional verification by making assumptions on the interaction between the connected interlockings in order to ignore them. By opposition, we formalize the existing connections (interfaces) defined in our interlockings through our compositional framework (OCRA) as they can't be ignored. In their research [Win01, Win12], Winter *et al.* propose to use abstract state machine to encode the application data before optimizing the ordering of the variables used in the BDD algorithm of NUSMV. The same approach was used in our first case study [BCL+15] but didn't permit to verify stations of significant size. Haxthausen *et al.* use BMC and induction in order to verify interlocking data with the RT-Tester tool-box [HPP14]. They also demonstrate how to model the sequential release process of the interlocking [HHP14]. The sequential release is used to increase the capacity of a railway network and is also implemented in our model whereas we only use BMC to verify liveness properties.

2 Compositional verification of interlockings

In this section, we explain why we need to use the compositional verification for application data of interlocking. We also provide the reader with the theoretical basis underlying the proof system implemented in the OCRA framework.

2.1 A justification for the compositional verification

In the railway domain, the norm [CEN02] defines the safety integrity level (SIL) for the development of safety critical systems and imposes to use the highest level (4) for the interlocking. One of the consequences is that the interlocking must ensure the computation of its outputs and internal states within a given period of time which in terms restricts the number of objects that it can control. For example, a SSI (Solid State Interlocking) [F. 13] can control maximum 64 points, 128 signals, 256 tack-circuits, and 256 routes. On the other hand, we want to diminish the impact of an interlocking failure on the train traffic. That's why, we wouldn't never control a significant size station with only one interlocking but we would use several of them.

All these arguments highlight the fact that an interlocking can't be seen as an isolated system. In fact the railway network must be considered as a graph of interconnected interlockings with borders and interfaces. A border is a boundary between two control areas whereas an interface is a connection point where interlockings exchange data. Those interfaces and the information that they carry **must guarantee the safety** of the railway traffic when trains are heading for a border.

2.2 Compositional verification

We use the OCRA compositional verification framework in order to verify a connected interlocking graph. Practically, OCRA provides: 1) a formal framework to specify a system made of a hierarchy of connected components 2) a formal language to define the properties to be verified on the system, and 3) a set of verification tools. In OCRA, the keyword **system** must be understood as a graph of interconnected interlockings (composite).

A component represents an interlocking instantiated by its SMV model (NUXMV) and is later called the leaf model. A component can also be a part of interlocking as it's sometimes possible to split an interlocking model into smaller ones in order to tackle the state space explosion problem (Section 4). The system and its components are described in Othello language proposed by Cimatti *et al.* [Cim13]. Othello allows for complex combinations of linear temporal operators, Boolean connectives, regular expressions, over terms referring to the variables of components ([CT12]). The contracts-refinement proof system for component-based systems was proposed by Cimatti *et al.* in the paper [CT15].

The verification of the whole system (graph of interlockings) takes place in three steps:

1. checking that the components contracts entail the system contract
2. checking that each component satisfies its contracts
3. checking that the environment of each component satisfies the assumption of that component

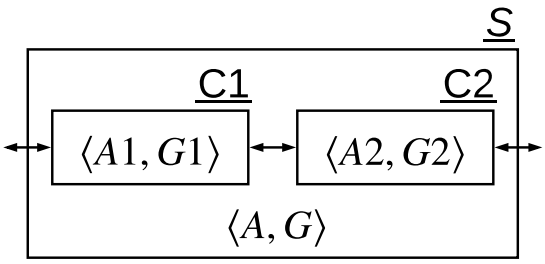

Figure 2.1: SYSTEM represented as a hierarchy of components (C1 and C2) and contracts.

The properties to be verified on a system (e.g., S in Figure 2.1) are expressed by means of the high-level contracts. A contract C for a system S is an assume-guarantee pair of (LTL Linear Temporal Logic) assertions $\langle A, G\rangle$ where $[\![A]\!]$ and $[\![G]\!]$ are sets of traces over the interfaces of S. A trace represents the observable part of a run of a component. It consists of the events and values of data ports (interface variables). The contract $\langle A, G\rangle$ applies to the system S and is refined by contract $\langle A1, G1\rangle$ for C1 and contract $\langle A2, G2\rangle$ for C2. **In the first step**, we verify that the component contracts entail the system contract (Eq. 2.1).

$$((\neg A1 \vee G1) \wedge (\neg A2 \vee G2)) \rightarrow (\neg A \vee G) \tag{2.1}$$

equa 2.1: Components contracts must entail the system contract.

In the second step, we verify that each component properly implements its contract. More formally we say that I is an implementation satisfying C iff $I \cap [\![A]\!] \subseteq [\![G]\!]$ ($I \models A \rightarrow G$). In our example, it means verifying that $I1 \cap [\![A1]\!] \subseteq [\![G1]\!]$ and $I2 \cap [\![A2]\!] \subseteq [\![G2]\!]$. $I1$ and $I2$ are respectively implementations for the C1 and C2 components. For the verification of the implementation, OCRA calls NUXMV.

In the last step, we must verify that the environment of each component satisfies its contract. More formally, we say that E is an environment satisfying C iff $E \subseteq [\![A]\!]$. For our example in Figure 2.1 it is verified by checking the validity of the following formulas 2.2:

$$\begin{array}{lcl} A \wedge (\neg A2 \vee G2) & \rightarrow & A1 \\ A \wedge (\neg A1 \vee G1) & \rightarrow & A2 \end{array} \tag{2.2}$$

equa 2.2: Verification of the environment of each component.

If the three steps of verification are valid, we say that the composition of *C1* and *C2* forming *S* satisfies the system property or more formally $I \models A \rightarrow G$. To be complete, we provide List. 2 which is an example of contracts and interfaces definition in Othello partially covering the toy example in Figure 3.2. The high level contract (line 6-8) that we want to prove on the system S is:

Train_from_XX_to_10D → !Train_from_YY_to_10D (line 9). 10D is a location at the interface between interlockings *XX* and *YY* where a collision might happen. The property states that **never** a train coming from *XX* shall head for 10D at the same time as a train coming from *YY*. This contract is refined by two contracts at the *XX* (line 26-28) and *YY* component level. Ud_10D_B and Ud_10D_A are two variables shared among the components *XX* and *YY*. The interfaces and the connections between the components are defined in lines 14 and 15. The variables Train_from_XX_to_10D, Ud_10D_A, and Ud_10D_B are declared in the leaf model of *XX* and allow the verification of the contract at component level (step 2). Our listing only shows the definition of the *XX* component.

```
1  COMPONENT Ath system
2  INTERFACE
3  OUTPUT PORT Train_from_XX_to_10D: boolean;
4  OUTPUT PORT Train_from_YY_to_10D: boolean;
5  -- #2 Top level contracts
6  CONTRACT Train_10D
7    assume: always TRUE;
8    guarantee: always (Train_from_XX_to_10D -> !Train_from_YY_to_10D
       );
9  -- #3 sub-components
10 REFINEMENT
11 SUB SSIXX  : XX;
12 SUB SSIYY  : YY;
13 -- #4 connections
14 CONNECTION SSIXX.Ud_10D_B := SSIYY.Ud_10D_B;
15 CONNECTION SSIYY.Ud_10D_A := SSIXX.Ud_10D_A;
16 -- #5 contracts refinements
17 CONTRACT Train_10D
18 REFINEDBY VIXLXX.Train_10D, VIXLYY.Train_10D;
19 -- #6 first component
20 COMPONENT XX
21 INTERFACE
22 INPUT  PORT Ud_10D_B: boolean;
23 OUTPUT PORT Ud_10D_A: boolean;
24 OUTPUT PORT Train_from_XX_to_10D: boolean;
25 -- #7 contract refinement at component level
26 CONTRACT Train_10D --
27   assume: always TRUE;
28   guarantee: always (Train_from_XX_to_10D -> (!Ud_10D_A & Ud_10D_B
       ));
```

Listing 2: Example of OCRA configuration file.

3 Catalogue of interfaces and contracts

In this section, we describe our catalogue of interface and contract definitions which supports the automatic production of the OCRA configuration file (e.g., List. 2) in order to perform the

compositional verification of graphs of interlockings. Our catalogue is summarized in Table 1 and holds five different types of interface.

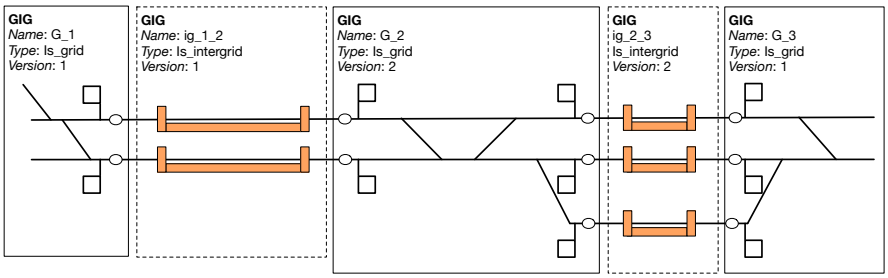

Figure 3.1: EXAMPLE of decomposition of a railway network into grids and inter-grids.

First, we define the notion of grid and inter-grid. A **grid** is a part of the railway network where absolute signals (route origin signal), points, and treadles (wheel sensors installed on the rail) are found. Three grids can be found in Figure 3.1: G_1, G_2, and G_3. The **inter-grids** are found between the grids and don't hold any points nor any absolute signals. In Figure 3.1, there are two inter-grids: ig_1_2 and ig_2_3. In order to avoid train head-on collisions in an inter-grid, we use a mutual exclusion mechanism called **directional locking** (first three entries in Table 1). To put it briefly, this mechanism ensures, as an example, that the interlocking controlling G_1 cannot send a train towards G_2 if the interlocking controlling G_2 is sending a train towards G_1 through ig_1_2 at the same time.

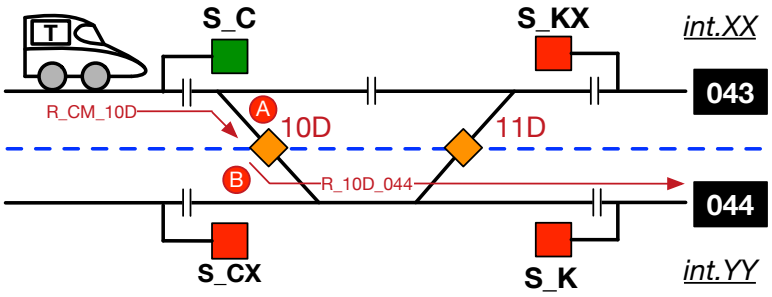

Figure 3.2: SECTIONING point at the border between interlocking XX and YY (second toy example).

Second in order to better understand what a contract materialize, we provide a second toy example of a grid controlled by two interlockings separated by the blue line: *int.XX* and *int.YY*. At the border between the interlockings, we find two sectioning points (orange diamonds): 10D and 11D. An itinerary from signal C to track 044 requires to command two routes: R_C_10D in int.XX and R_10D_044 in int.YY. An itinerary can involve several routes in different interlockings to form an end to end path in a station. The sectioning point has two ends: end A on its left side and end B on its right side. This translates in the application data by two variables: U_10D_A and U_10D_B. The first one is controlled by int.XX and the second one by int.YY.

These variables are used to formalize a contract between the components representing int.XX and int.YY guaranteeing that the two interlockings shall never command a route towards the shared sectioning point 10D (fourth entry in Table 1).

Third, we apply conventions to the OCRA configuration files. Firstly we restrict the hierarchy of the connected components to two layers: the **system** (station) and the components (interlockings) as in Figure 2.1. This saves forwarding the connections between different layers (system $\rightarrow$ component $\rightarrow$ sub-component). After we define the high level contracts as a "no collision" property that, for our toy example, would be written as: **!(Train_from_XX_to_10D & Train_from_YY_to_10D)**. The generic form is Train_from_CCC_to_III where CCC is the component and III the interface point. The contracts at component level (column 3 in Table 1) refine the top level contract and are verified on the SMV model. The generic form of the interface is defined as TTT_SSS_III where TTT is the type of interface, SSS the side/direction (A, B, or signal), and III the interface name. We argue that our table covers all possible cases of interface used in the Belgian signalling principles.

Table 1: CATALOGUE of interfaces and contracts.

Type	Variable	Contract
BSI	U_BSI[AB]_III	$Train\_from\_CCC1\_to\_III \rightarrow (\neg U\_BSIA\_III \wedge U\_BSIB\_III)$ $Train\_from\_CCC2\_to\_III \rightarrow (\neg U\_BSIB\_III \wedge U\_BSIA\_III)$
BSP	L_BSP[AB]_III	$Train\_from\_CCC1\_to\_III \rightarrow (\neg L\_BSPA\_III \wedge L\_BSPB\_III)$ $Train\_from\_CCC2\_to\_III \rightarrow (\neg L\_BSPB\_III \wedge L\_BSPA\_III)$
BSRM	L_VAS_III, L_VSD_III	$Train\_from\_CCC\_to\_III \rightarrow (\neg L\_VAS\_III \wedge L\_VSD\_III)$
Spoint	U_10D_[AB]	$Train\_from\_XX\_to\_10D \rightarrow (\neg U\_10D\_A \wedge U\_10D\_B)$ $Train\_from\_YY\_to\_10D \rightarrow (\neg U\_10D\_B \wedge U\_10D\_A)$
Dpoint	U_RSU_III_[AB]	$Train\_from\_CCC1\_to\_III \rightarrow (\neg U\_RSU\_III\_A \wedge U\_RSU\_III\_B)$ $Train\_from\_CCC2\_to\_III \rightarrow (\neg U\_RSU\_III\_B \wedge U\_RSU\_III\_A)$

The first three entries of Table1 apply to the directional locking (i.e.,blocking) that comes in three varieties : BSI, BSP, and BSRM. The BSI applies to the platforms and is identified by the variable U_BSI[AB]_III where [AB] gives the direction ($\rightarrow$ or $\leftarrow$) and III the name of the platform. The BSP applies to the tracks inside a grid where a level crossing is encountered for example. It's identified by the variable U_BSP[AB]_III where [AB] gives the direction and III the name of the track. The BSRM is used between two stations. Two variables are used: L_VAS and L_VSD. As explained previously these three interfaces are used in the inter-grids. The last

two types of contracts are related the **Sectioning** points (Spoint as explained in the toy example) and the distribution point (Dpoint). The **Distribution** points are used for rail yards. Their usage is very similar to that of the Spoint except that the variable is U_RSU. Our catalogue is based on the Belgian signalling principles but could be extended for other countries (e.g., France). It has the advantage of being readable by the railway experts.

4 An algorithm to split the interlocking topology

The concept of compositional verification relies on 1) the definition of the components/interfaces and 2) the definition of contracts connecting the components through their interfaces. We define an algorithm that takes a railway grid data (as represented by SSI) and decomposes into components and interfaces with corresponding contracts.

The algorithm relies on a collection of route objects holding their characteristics like their name (unique for an area), the origin signal, and a finite set of elements (resources used by the route). Different types of elements are defined, such as: points, track-circuits, signals, and locking variables. Those types are categorized into two subtypes: the internal resources used to set the route (e.g., points) and the interface elements which are typically encountered at the interface between components (e.g., directional locking). For the route R_C_043 of Figure 1.1, the internal resources are {P_01A, P_02B,U_IR_01A, U_IR_02B, S_C, T_01, T_02} whereas {U_BSPA_043} is an external element preventing collision between the routes in opposite direction leading to the same destination point: Track 043. Two routes are said to be in opposite direction if one goes from left to right and the second one from right to left.

The generation of the components can be automated following alg. 4.1 which takes the list of routes belonging to the grids of the station as input (line 1). The intermediate output *comps* has a map from component's name $\rightarrow$ list of routes. For example, one entry of the resulting map could be: "component_236" $\rightarrow$ List{R_C_043, R_C_044, R_CX_043, R_CX_044}. Another intermediate map is used to map the routes to their component (e.g., *compName* entry: "R_C_043" $\rightarrow$ "component_236"). The core of the algorithm is a nested loop (line 2-3) where the lists of routes are compared pairwise (*rt1* and *rt2* - line 6). The method *Elements(rt)* returns the **internal** elements of the route as defined previously. If the *rt1* and *rt2* are disjoint (line 19), the two routes are inserted into separated components, a new component is created if needed. If the element lists are not disjoint and if the two routes already belong to two different components, the two components are **merged** (line 11). Otherwise if either *rt1* or *rt2* already belongs to a component, the other route is added to its routes list (lines 13-14). Finally if neither *rt1* nor *rt2* belongs to a component, a new component is created and both routes are added to it (lines 15-18).

In the second part of the algorithm, we process *comps* in order to produce the configuration file for OCRA (lines 28-34). For each component (line 29) of *comps* and for each of their routes (line 30), we extract the interfaces in the elements list of the route. The interfaces are easily extracted as they were classified as interface elements. We attach a contract to each interface based on the mapping interface $\leftrightarrow$ contract defined in our catalogue. For the example in Figure 1.1, the algorithm would discover U_BSPA_043 as an interface of type *BSP* for route R_C_043. The routes of a component can have different interfaces. For the example in Figure 1.1, the algorithm would discover the interface U_BSPA_044 for the route R_C_044. In the component definition,

Algorithm 4.1: Retrieve components and produce OCRA config. file

```
1  Data: RT route list
2  forall rt1 ∈ RT do
3      forall rt2 ∈ RT if rt2 ≠ rt1 do
4          comp1 = compName[rt1]
5          comp2 = compName[rt2]
6          if Elements(rt1) ∩ Elements(rt2) ≠ ∅ then
7              if comp1 ≠ ⊥ AND comp2 ≠ ⊥ then
8                  if comp1 ≠ comp2 then
9                      comp = new component
10                     routes = comps[comp1] ∪ comps[comp2]
11                     comps = comps -comp1 - comp2 + comp → routes
12                     compName[rt1] = compName[rt2] = comp
13             else if comp1 ≠ ⊥ then
                   comps[comp1] = comps[comp1] + rt2
14             else if comp2 ≠ ⊥ then
                   comps[comp2] = comps[comp2] + rt1
15             else Create new component
16                 comp = new component
17                 comps[comp] = {rt1, rt2}
18                 compName[rt1] = compName[rt2] = comp
19         else Add rt1 and rt2 in components if needed
20             if comp1 ≠ ⊥ then
21                 comp = new component name
22                 comps[comp] = {rt1}
23                 compName[rt1] = comp
24             if comp2 ≠ ⊥ then
25                 comp = new component name
26                 comps[comp] = {rt1}
27                 compName[rt1] = comp
28 Data: comps
29 forall comp ∈ comps do
30     forall rt ∈ comp do
31         interfaces = interfaces + externalElements[rt]
32         contracts = contracts + contractCatalogue[interfaces]
33     PrintComponent(Name, interfaces, contracts)
34 PrintHightLevelContracts
```

we print its name, its list of interfaces and its contracts (line 33). We also collect all the interfaces and contracts of all the components (*Allcontracts* and *Allinterfaces*) and write the contract refinement at system level (line 34).

After completion of Alg.4.1, we get a file written in Othello usable by OCRA in order to perform the compositional verification of our graph of interlockings. Another benefit of our algorithm is that it can split a large interlocking into smaller components. This means that we can perform the verification of a large interlocking though the verification of smaller components by applying the compositional verification. The splitting doesn't happen for every interlocking but is more likely to happen on larger ones where the state space explosion problem is a major obstacle. An example of splitting is provided in our case study.

5 Verification strategy and case study

In this section, we briefly describe the generic complete verification process applied to the composition of interlocking components. We explain the results obtained after applying our algorithm to the station of Ath on the Belgium railway.

5.1 Verification method

The property verified in the framework of this paper is the **"no-collision"** property. In order to verify a network of interlockings, we proceed in two steps. First we verify that no collision can happen at the interfaces between the components (i.e., interlockings or part of interlockings). This is illustrated by the *I* between *C1* and *C2* of Figure 5.1. Second we verify that no collision can happen inside each component *C1* and *C2*.

For the first step, we specify the "no-collision" property by means of a contract: "No train coming from *C1* shall reach a position corresponding to the interface between *C1* and *C2* at the same time as a train coming from *C2*. This high level contract (i.e., system level) must be refined by component contracts at *C1* and *C2* level. The refinement is verified by means of OCRA whereas the implementation is verified at the leaf level (i.e., implementation of each component in SMV). Each contract is taken from the catalogue defined in Section 3 and automatically identified by our algorithm. Second, in order to verify the "no-collision" property inside each component, the SMV interlocking modules are composed with two different instances of the same train module (e.g., $C1 \| ins11 \| ins12$ for *C1*). The train is also a SMV module that is configured based on the topology of the station. When an origin signal of a route turns green (a.k.a.permissive aspect), the train starts in front of that signal and follows the topology of the railway by taking the points according to their commanded position. The train also actuates the track-circuits (a.k.a.track segments) which trigger the releasing of the route resources in a predefined sequence. The train module is extracted from the GRAFFITI XML file that is a fork from the railML standard [rai15] developed by the Belgian railways. This file holds the topology of the station as well as a description of all its resources (e.g., points, signals, ...).

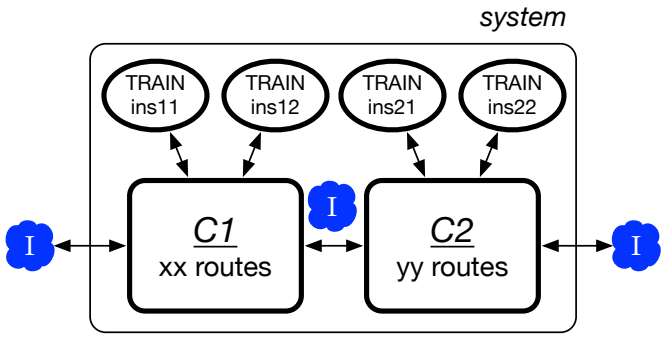

Figure 5.1: GENERIC case of network of interlockings composed with train simulation.

5.2 Case study - Application of the verification process

Our case study comes from a fairly large station of the Belgian railway: the Ath station. This station is controlled by three VIXL: fty_7, fty_8, and fty_9 with respectively 48, 68, and 34 routes (Fig 5.2). A VIXL is a virtual interlocking running in a central interlocking (a.k.a.CIXL). **fty** is the telegraphic identification of the station.

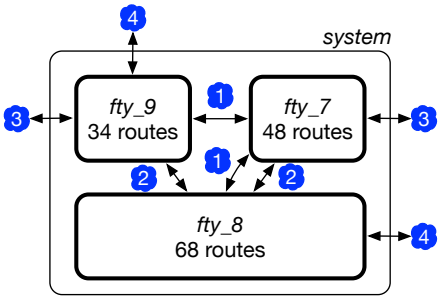

Figure 5.2: ATH station - Components breakdown structure according corresponding to the interlocking VIXL structure.

Figure 5.3 shows the result of the splitting (decomposition) with our algorithm. **fty_9** could not get decomposed further and remains one monolithic model. Indeed the algorithm could not identify any interface defined in our catalogue. **fty_8** was decomposed into five components (sub-interlocking modules): A, B, C, D, and E. However, we decide to keep two groups: A+B and C+D+E. This reduces the number of interfaces to be defined while allowing a substantial benefit in terms of the size of the models to be verified. The two sub-models are isolated from the original model of interlocking **fty_8**: one with 22 routes and a second one with 46 routes. Finally **fty_7** was split into two components. However, as the size of component fty_7B was not significant, the verification was achieved on the complete model.

Table 2 summarizes the performance of the two steps process of verification of our case study. The computer that we used was a HP Elite i5-6200U - 2.3GHz - 4 processors - 8GB running Xubuntu 18.04. The verification of the refinement of the 12 contracts (e.g., lines 18-19 of List. 2)

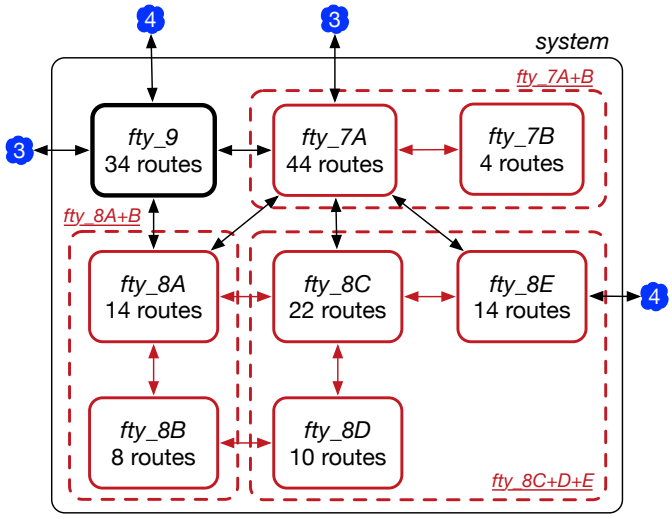

Figure 5.3: ATH station - Components breakdown structure after refinement by the splitting algorithm.

was performed in one run by OCRA in a matter of seconds. The verification of the proper implementation of the contracts in the SMV leaf components was achieved by NUXMV. The verification of the "no-collision" property on the leaf components (step 2) was performed by four concurrent instances of NUXMV by means of the **PDR/ic3** algorithm [Bra11b, Bra12, Bra11a, SB11]. Prior to the verification, we validated our model. First we seeded faults inside the application data and checked that the "no-collision" property got violated. Second we verify some liveness properties like: every route can be commanded and ran through by the train or each track-circuit can be occupied by a train. Examples of traces can be provided on demand.

Table 2: DURATION of the verification process.

Steps	Verification		Duration
1.1	System Contracts refinement	12 contracts	65.5 s
1.2	Contracts implementation for fty_7A+B	5 contracts	24.6 s
	Contracts implementation for fty_8A+B	8 contracts	79.2 s.
	Contracts implementation for fty_8C+D+E	8 contracts	76.7 s.
	Contracts implementation for fty_9	3 contracts	10.1 s.
2	No collision in fty_7A+B		42.85 h
	No collision in fty_8A+B		8.64 h
	No collision in fty_8C+D+E		37.61 h
	No collision in fty_9		36.21 h

As can be seen from the result table, the verification of the leaf SMV model is the hardest as it can take up to 43 hours to prove the "no collision" property in the application data of an interlocking with 48 routes (fty_7 = 586 Boolean variables). Before splitting the interlocking fty_8,

we tried to verify the "no collision" property on the monolithic model with NUXMV but it had not terminated after three (3) days. The decomposition with our algorithm allowed to performed its verification at the price of defining two components with their interfaces and contracts. As this process is automatic and quick ($\pm$ 3 minutes) thanks to our algorithm and catalogue, we think that it's a solution for the verification of fairly big stations.

6 Conclusions

Medium and large stations are controlled by a network of interlockings due to the limitations of the current technology and due to some availability constraints. In this article, we have shown that the compositional verification is a perfect fit for the formal verification of such a network. This principle can be further used to split the interlocking entity thereby allowing to tackle the state space explosion problem.

Our contribution is twofold. First we proposed an algorithm that can automatically split an interlocking entity into smaller entities called components. Second we proposed a catalogue containing an exhaustive list of interfaces/contacts allowing the compositional verification of all our graph of interlockings. This catalogue holds all the templates of the formal definitions needed to declare the interfaces and contracts binding the components. Our algorithm thus produces the components, identifies the interfaces with other components, and infers the contracts ruling the interfaces. This output can then be directly used by the OCRA tool to perform the compositional verification of the system (i.e., group of components/interlocking).

In our case study, we demonstrated the benefits of the use of the algorithm on a realistic size station of the Belgian railways controlled by three interlockings. We developed our two-steps process and showed that it allows to get the prove of the "no-collision" property for the 150 routes station.

References

[A. 17] A. Fantechi and A. E. Haxthausen and M. B. R. Nielsen. Model Checking Geographically Distributed Interlocking Systems Using UMC. In *2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. Pp. 278–286. March 2017.
doi:10.1109/PDP.2017.66

[BCL+15] S. Busard, Q. Cappart, C. Limbrée, C. Pecheur, P. Schaus. Verification of railway interlocking systems. In *Proceedings 4th International Workshop on Engineering Safety and Security Systems, ESSS 2015, Oslo, Norway, June 22, 2015*. Pp. 19–31. 2015.
doi:10.4204/EPTCS.184.2
http://dx.doi.org/10.4204/EPTCS.184.2

[BP13] S. Busard, C. Pecheur. PyNuSMV: NuSMV as a Python Library. In Brat et al. (eds.), *Nasa Formal Methods 2013*. LNCS 7871, pp. 453–458. Springer-Verlag, 2013.

[Bra11a] A. R. Bradley. SAT-Based Model Checking without Unrolling. In *VMCAI*. LNCS 6538, pp. 70–87. Springer, 2011.

[Bra11b] A. Bradley. SAT-Based Model Checking without Unrolling. In Jhala and Schmidt (eds.), *Verification, Model Checking, and Abstract Interpretation*. Lecture Notes in Computer Science 6538, pp. 70–87. Springer Berlin Heidelberg, 2011.
doi:10.1007/978-3-642-18275-4_7
http://dx.doi.org/10.1007/978-3-642-18275-4_7

[Bra12] A. R. Bradley. Understanding IC3. In *Theory and Applications of Satisfiability Testing – SAT 2012*. Pp. 1–14. Springer Science + Business Media, 2012.
doi:10.1007/978-3-642-31612-8_1
http://dx.doi.org/10.1007/978-3-642-31612-8_1

[CDT13] A. Cimatti, M. Dorigatti, S. Tonetta. OCRA: A tool for checking the refinement of temporal contracts. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. Institute of Electrical & Electronics Engineers (IEEE), nov 2013.
doi:10.1109/ase.2013.6693137
http://dx.doi.org/10.1109/ase.2013.6693137

[CEN02] CENELEC. IEC61508 - Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems. 2002.

[Cim13] Cimatti, Alessandro and Roveri, Marco and Susi, Angelo and Tonetta, Stefano. Validation of Requirements for Hybrid Systems: A Formal Approach. *ACM Trans. Softw. Eng. Methodol.* 21(4):22:1–22:34, Feb. 2013.
doi:10.1145/2377656.2377659
http://doi.acm.org/10.1145/2377656.2377659

[CT12] A. Cimatti, S. Tonetta. A Property-Based Proof System for Contract-Based Design. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*. Institute of Electrical & Electronics Engineers (IEEE), sep 2012.
doi:10.1109/seaa.2012.68
http://dx.doi.org/10.1109/seaa.2012.68

[CT15] A. Cimatti, S. Tonetta. Contracts-refinement proof system for component-based embedded systems. *Science of Computer Programming* 97:333–348, jan 2015.
doi:10.1016/j.scico.2014.06.011
http://dx.doi.org/10.1016/j.scico.2014.06.011

[F. 13] F. M. Spowart. SSI – Solid State Interlocking : Basic guide. Brochure, 2013.
http://nremployee.yolasite.com/resources/SSI%20basic%20system%20notes%20(PDF).pdf

[FFM12] A. Fantechi, W. Fokkink, A. Morzenti. *Some Trends in Formal Methods Applications to Railway Signaling*. Pp. 61–84. John Wiley & Sons, Inc., 2012.

doi:10.1002/9781118459898.ch4
http://dx.doi.org/10.1002/9781118459898.ch4

[HHP14] L. V. Hong, A. E. Haxthausen, J. Peleska. Formal Modeling and Verification of Interlocking Systems Featuring Sequential Release. In *Formal Techniques for Safety-Critical Systems - Third International Workshop, FTSCS 2014, Luxembourg, November 6-7, 2014. Revised Selected Papers*. Pp. 223–238. 2014.
doi:10.1007/978-3-319-17581-2_15
https://doi.org/10.1007/978-3-319-17581-2_15

[HPP14] A. E. Haxthausen, J. Peleska, R. Pinger. Applied Bounded Model Checking for Interlocking System Designs. In *Revised Selected Papers of the SEFM 2013 Collocated Workshops on Software Engineering and Formal Methods - Volume 8368*. Pp. 205–220. Springer-Verlag New York, Inc., New York, NY, USA, 2014.
doi:10.1007/978-3-319-05032-4_16
http://dx.doi.org/10.1007/978-3-319-05032-4_16

[Lim16] Limbrée, Christophe and Cappart, Quentin and Pecheur, Charles and Tonetta, Stefano. Verification of railway interlocking-compositional approach with OCRA. In *International Conference on Reliability, Safety and Security of Railway Systems*. Pp. 134–149. 2016.

[ocr18] OCRA Tool - Home Page. May 2018.
https://ocra.fbk.eu/

[rai15] The XML-Interface for Railway Applications. March 2015.
http://www.railml.org

[Rob14] Roberto Cavada and Alessandro Cimatti and Michele Dorigatti and Alberto Griggio and Alessandro Mariotti and Andrea Micheli and Sergio Mover and Marco Roveri and Stefano Tonetta. The nuXmv Symbolic Model Checker. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*. Pp. 334–342. 2014.
doi:10.1007/978-3-319-08867-9_22
http://dx.doi.org/10.1007/978-3-319-08867-9_22

[SB11] F. Somenzi, A. R. Bradley. IC3: Where Monolithic and Incremental Meet. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*. FMCAD '11, pp. 3–8. FMCAD Inc, Austin, TX, 2011.
http://dl.acm.org/citation.cfm?id=2157654.2157657

[Win01] K. Winter. *Model Checking Abstract State Machines*. It, Von der Fakultat IV - Elektrotechnik und Informatik der Technischen Universitat Berlin, July 2001.

[Win12] K. Winter. Optimising Ordering Strategies for Symbolic Model Checking of Railway Interlockings. In *Proceedings of the 5th International Conference on Leveraging Applications of Formal Methods, Verification and Validation: Applications and Case Studies*

- *Volume Part II*. ISoLA'12, pp. 246–260. Springer-Verlag, Berlin, Heidelberg, 2012.
doi:10.1007/978-3-642-34032-1_24
http://dx.doi.org/10.1007/978-3-642-34032-1_24